THE CHIPS TO SYSTEMS CONFERENCE
SHAPING THE NEXT GENERATION OF ELECTRONICS

JUNE 23-27, 2024
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

Jiaxi Jiang¹, Lancheng Zou¹, Wenqian Zhao¹, Zhuolun He¹, Tinghuan Chen², Bei Yu¹

¹Chinese University of Hong Kong
²Chinese University of Hong Kong, Shenzhen
Outline

1 Introduction

2 Algorithm: Iterative parallel sweepline algorithm

3 Experiments
Non-Manhattan geometry: X or any-angle shapes used in routing\textsuperscript{12}. Design rule checking (DRC): Verify a design’s layout against geometric rules.

The increasing size of package designs may impact the efficiency of DRC.

---


\textsuperscript{2}T. Chen \textit{et al.}, “TRouter: Thermal-driven PCB Routing via Non-Local Crisscross Attention Networks”, \textit{IEEE TCAD}, 2023.
PDRC deals with package/PCB designs featuring non-Manhattan geometry, expanding layout shapes and employing GPU-accelerated geometric intersection algorithms to finish design rule checking.
From a computational geometry perspective

Previous researches use computational geometry tools to tackle design rule checking\(^3\)\(^4\)\(^5\).

**partition** the input space and data

**partition space** Quad-tree recursively partitions the layout space.

**partition data** R-tree partitions layout objects, minimizing coverage and overlap among subnodes.

**spatial order** The spatial order of line segments is determined by the coordinates of their intersection points with the sweepline.

---


X-Check$^6$ utilizes the y-coordinates of horizontal segments to establish order, akin to partitioning the layout along the y-axis.

A rough comparison

sweepline: The efficiency of the sweepline algorithm is attained by limiting the search space to immediate neighbors.

bounding box: The partitioning strategy employs bounding boxes to approximate diagonal lines, creating empty spaces that reduce pruning efficiency.
Uniqueness Any vertical (horizontal) line through the objects intersects the object exactly once.

Orderliness For objects intersecting the line, they can have a total order.

Computability Given two objects, it is possible to compute the intersection point.

All convex polygons exhibit Uniqueness property.

To have an order, points are preferred over segments.
Issues come from decomposition

Decomposition prevents the detection of inclusion.

Reassemble the decomposed shapes by connecting the points.

Evaluate if any overlap exists between line segments on the same sweepline to perform an inclusion check.

By sorting and scanning the endpoints of line segments, we can efficiently detect overlapping segments with a single scan.
Iterative parallel sweepline algorithm

event Starting or ending points of segments, or intersection points.

Sweepline processes all events in order, maintaining the order of intersecting objects with the sweepline using a binary tree, and utilizing a priority queue to maintain the order of all events.

Observation\textsuperscript{7} that concurrently processing events yields the same results as sequential execution.

Iterative parallel sweepline algorithm

- The **initial sweeplines** are generated based on the x-coordinates of line segments.
• The **initial sweeplines** are generated based on the x-coordinates of line segments
• While **iterative sweeplines** are produced from intersection points.
Efficiently determining sets of segments intersected by the sweepline at each position requires **stab queries** for line segments projected along the x-axis.
• Efficiently determining sets of segments intersected by the sweepline at each position requires **stab queries** for line segments projected along the $x$-axis.

• We assign distinct **hierarchical labels** to line segments based on their lengths, allowing for the organization of segments from each hierarchy into separate interval lists.
Iterative parallel sweepline algorithm

For each event’s position,

- **Label** Identify the left and right boundaries in interval lists \( \mathcal{L} \).
- **Merge** Merge the segment sequence as \( S \) from \( \mathcal{L} \).
- **Sort** Sort segments in \( S \) based on the \( y \)-coordinates where they intersect with the event sweepline.
- **Check** Scan segments and update intersection events.
- **Iterative Check** Choose merged segments, sort and scan.
Complexity analysis

**Label** Binary searches across $k$ interval lists with a total of $n$ intervals yield a depth of $O(\log \frac{n}{\omega_k})$. The total work required is at most $O(k \log \frac{n}{k})$ for each position.

**Merge** In the worst-case scenario, the depth reaches $O(\sqrt{n})$, leading to a total work of $O(p \sqrt{n})$ across $p$ positions.

**Sort** Parallel radix sorting, when applied to each list, achieves a depth of $O(d \log n)$, where $d$ denotes the number of digits, with the total work $O(dp \sqrt{n})$.

**Check** Each position requires a single scan through the interval list for inclusion and intersection checks, leading to a depth of $O(\sqrt{n})$ and a work of $O(p \sqrt{n})$.

**Iterative Check** When few intersections occur, the “Iterative Check” stage requires $O(1)$ iterations.
Benchmarks

Our benchmarks include some industrial PCB designs.

**Table:** The statistics of our benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#C</th>
<th>#P</th>
<th>#N</th>
</tr>
</thead>
<tbody>
<tr>
<td>xc7z020_t</td>
<td>443</td>
<td>1737</td>
<td>428</td>
</tr>
<tr>
<td>xc7z020_b</td>
<td>572</td>
<td>1390</td>
<td>383</td>
</tr>
<tr>
<td>xc7z030_t</td>
<td>447</td>
<td>1936</td>
<td>442</td>
</tr>
<tr>
<td>xc7z030_b</td>
<td>653</td>
<td>1539</td>
<td>416</td>
</tr>
<tr>
<td>hs3690_t</td>
<td>910</td>
<td>3529</td>
<td>998</td>
</tr>
<tr>
<td>hs3690_b</td>
<td>656</td>
<td>1878</td>
<td>496</td>
</tr>
</tbody>
</table>

The statistics of our benchmarks are listed on Table 1, where #C, #P, #N denote the numbers of components, pads and nets respectively. Additionally, we replicate our largest benchmark, by factors of 4, 8, and 16.
## Runtime comparisons

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>#Segments</th>
<th>Klayout flat</th>
<th>Klayout deep</th>
<th>Klayout tile</th>
<th>R-tree boost</th>
<th>PDRC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RT</td>
<td>Ratio</td>
<td>RT</td>
<td>Ratio</td>
<td>RT</td>
</tr>
<tr>
<td>xc7z020_t</td>
<td>39368</td>
<td>301</td>
<td>43.0×</td>
<td>272</td>
<td>38.9×</td>
<td>146</td>
</tr>
<tr>
<td>xc7z020_b</td>
<td>18014</td>
<td>232</td>
<td>77.3×</td>
<td>173</td>
<td>57.7×</td>
<td>65</td>
</tr>
<tr>
<td>xc7z030_t</td>
<td>45972</td>
<td>273</td>
<td>91.0×</td>
<td>275</td>
<td>91.7×</td>
<td>108</td>
</tr>
<tr>
<td>xc7z030_b</td>
<td>19500</td>
<td>235</td>
<td>78.3×</td>
<td>189</td>
<td>63.0×</td>
<td>69</td>
</tr>
<tr>
<td>hs3690_t</td>
<td>68604</td>
<td>825</td>
<td>165.0×</td>
<td>705</td>
<td>141.0×</td>
<td>331</td>
</tr>
<tr>
<td>hs3690_b</td>
<td>35082</td>
<td>452</td>
<td>150.7×</td>
<td>571</td>
<td>190.3×</td>
<td>192</td>
</tr>
<tr>
<td>4hs3690_t</td>
<td>274416</td>
<td>3334</td>
<td>222.3×</td>
<td>2772</td>
<td>184.8×</td>
<td>569</td>
</tr>
<tr>
<td>4hs3690_b</td>
<td>140328</td>
<td>1849</td>
<td>205.4×</td>
<td>2240</td>
<td>248.9×</td>
<td>358</td>
</tr>
<tr>
<td>8hs3690_t</td>
<td>548832</td>
<td>6721</td>
<td>268.8×</td>
<td>5636</td>
<td>225.4×</td>
<td>1064</td>
</tr>
<tr>
<td>8hs3690_b</td>
<td>280656</td>
<td>3731</td>
<td>186.6×</td>
<td>4445</td>
<td>222.3×</td>
<td>612</td>
</tr>
<tr>
<td>16hs3690_t</td>
<td>1097664</td>
<td>13470</td>
<td>244.9×</td>
<td>11274</td>
<td>205.0×</td>
<td>1996</td>
</tr>
<tr>
<td>16hs3690_b</td>
<td>561312</td>
<td>7505</td>
<td>220.7×</td>
<td>8929</td>
<td>262.6×</td>
<td>1136</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>143.0×</td>
<td>136.8×</td>
<td>35.3×</td>
<td>51.2×</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** Runtime (ms) comparisons of design rule checking(spacing).
The construction of **hierarchical interval lists** ("HIL") and **sweepline statuses** ("Build" and "Sort") account for the majority of the time. Owing to the limited number of intersection checks needed by sweepline algorithms, **Check** accounts for a small portion of the total runtime.
We’ve implemented an iterative parallel sweepline algorithm optimized for GPUs using position-level parallelism, and have efficiently conducted interval stabbing queries using hierarchical interval lists.

In the future, we plan to develop more work-depth-efficient algorithms and GPU-based data structures.
THANK YOU!